ELECTRICAL AND COMPUTER ENGINEERING
TELECOMMUNICATIONS DIVISION
Technical University of Crete

Applications of distributed inference in ambiently powered wireless sensor networks

A Thesis submitted by

Georgios Perakis

for the degree of

ELECTRICAL AND COMPUTER ENGINEERING

THESIS COMMITTEE

Professor Aggelos Bletsas, Thesis Supervisor

Professor Aikaterini Mania

Professor Thrasyvoulos Spiropoulos

Abstract

This thesis contains applications utilizing wireless sensor networks powered solely by the environment, using small, credit card-sized solar panels. The network contains a number of inexpensive terminals with transmission power of 10.4 dBm and token-ring medium access, capable of distributed in-network inference. An outdoor demonstration using loopy belief propagation and two indoor demonstrations using average consensus were developed and deployed, with this wireless network. The indoor demonstration includes two different versions; a centralised version focusing on robustness and a distributed counterpart focusing on available range. The indoor demonstrations calculate the average temperature in a closed space and the outdoor demonstration measures and estimates the soil moisture on an agricultural field; specific provisions are given so that soil moisture is estimated in locations where the sensors are broken or simply not reporting due to power outage. Distributed measurements and estimation for the outdoor demo required approximately 6 minutes of message passing between ambiently powered nodes; for the indoor demo, that time was approximately 3 minutes. It was found that distributed in-network inference with resource constrained, ambiently-powered wireless terminals is possible, at the expense of increased overall delay. In addition, it was found that distributed operation demands robust time synchronization among the terminals. Future work will focus on distributed time synchronization and other medium access control schemes for resource-constrained WSNs.

Thesis Supervisor: Professor Aggelos Bletsas

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisor for his invaluable guidance, encouragement, and unwavering support throughout the entire research and writing process. His expertise, patience, and inspiration have been critical to the successful completion of this thesis.

I would also like to thank Apostolakis Georgios for his valuable collaboration and contributions to this project. I am also grateful to Vougioukas Georgios, the post-PHD student, who provided valuable insights and assistance in the completion of this work.

Finally, I am deeply grateful to my family and Marilena for their constant support, encouragement, and patience throughout this challenging journey. Their unwavering support and understanding have been a source of motivation and inspiration for me. Vamolo ...

Contents

Abstract **i**

Acknowledgements **ii**

1 Introduction **1**
1.1 Wireless Sensor Networks . . . 1
1.2 Ambiently Powered Distributed Network . . . 1
1.3 Backscatter Radio . . . 2
1.4 Outline . . . 2
1.5 Related work . . . 2

2 Embedded Hardware **4**
2.1 Embedded Hardware . . . 4
2.1.1 Specifications . . . 4
2.2 Scatter Tag . . . 8
2.2.1 Scatter Board . . . 9
2.2.2 Power Board . . . 10
2.2.3 Sensor Board . . . 10
2.3 Solar Energy . . . 12
2.3.1 Solar Harvester . . . 12
2.3.2 Solar Irradiance . . . 13
2.4 Peltier . . . 14

3 Embedded Networking **15**
3.1 Networking . . . 15
3.1.1 Radio configuration . . . 15
3.1.2 Network Layer . . . 16
3.1.3 Baton Protocol . . . 16
3.1.4 MAC Address Protocol . . . 16
3.1.5 Backscatter Radio . . . 16
3.2 Non Volatile Memory (NVM) . . . 18
3.3 Real Time Clock and Sleeptimer . . . 18
3.4 Stack . . . 18

4 Wireless Sensor Networks as Inference Platforms **20**
4.1 Algorithm 1: Loopy Belief Propagation . . . 20
4.1.1 Loopy Belief Propagation . . . 20
4.1.2 Iterative Proportional Fitting . . . 22

4.1.3 Edge Potential Functions Computation 23
4.2 Algorithm 2: Average Consensus 23

5 Implementation of the WSNs **25**
5.1 Distributed Measurement and Estimation System for Agriculture 25
5.1.1 Sensor Measurement 25
5.1.2 Implementation 28
5.1.3 Timers 29
5.1.4 Radio Configuration 30
5.1.5 Packet Configuration 30
5.1.6 Command Line Interface 31
5.1.7 Initial Setup 31
5.1.8 Configuration 31
5.1.9 IPF Implementation 32
5.1.10 User Interface 32
5.1.11 Results 35
5.2 Distributed Averaging System for Handling Indoor Temperature 38
5.2.1 Centralized Variation 38
5.2.2 Distributed Variation 41

6 Conclusions **43**
6.1 Conclusion 43
6.2 Future Work 43

Bibliography **45**

List of Abbreviations

WSNs	**W**ireless **S**ensor **N**etworks
RGB	**R**ed **G**reen **B**lue
SoC	**S**ystem **o**n **C**hip
EM	**E**nergy **M**ode
USB	**U**niversal **S**erial **B**us
SPI	**S**erial **P**eripheral **I**nterface
UART	**U**niversal **A**synchronous **R**eceiver **T**ransmitter
I2C	**I**nter-Integrated **C**ircuit
GPIO	**G**eneral **P**urpose **I**nput - **O**utput
RF	**R**adio **F**requency
LED	**L**ight **E**mitting **D**iode
UV	**U**ltra **V**iolet
HVAC	**H**eating **V**entilation and **A**ir **C**onditioning
TUC	**T**echnical **U**niversity of **C**rete
ADC	**A**nalog to **D**igital **C**onverter
RTC	**R**eal **T**ime **C**lock
NiMh	**Ni**ckel **M**etal **H**ybrid
RAIL	**R**adio **A**bstraction **I**nterface **L**ayer
GFSK	**G**aussian **F**requency **S**hift **K**eying
NVM	**N**on **V**olatile **M**emory
SDK	**S**oftware **D**evelopment **K**it
FSM	**F**inite **S**tate **M**achine
LIFO	**L**ast **I**n **F**irst **Out**
API	**A**ppication **P**rogramming **I**nterface
FIFO	**F**irst **I**n **F**irst **Out**
LBP	**L**oopy **B**elief **P**ropagation
MRF	**M**arkov **R**andom **F**ield
IPF	**I**terative **P**roportional **F**itting
DMESA	**D**istributed **M**easurement and **E**stimation **S**ystem for **A**griculture
UI	**U**ser **I**nterface
CLI	**C**ommand **L**ine **I**nterface
DASHIT	**D**istributed **A**veraging **S**ystem for **H**andling **I**ndoor **T**emperature
MAC	**M**edium **A**ccess **C**ontrol
VWC	**V**olumetric **W**ater **C**ontent

Introduction

1.1 Wireless Sensor Networks

Wireless sensor networks (WSNs) are a type of network that integrates tiny sensing modules with communication capability, to monitor and act in various scenarios. Due to their versatility and potential for use in various applications, WSNs have the ability to revolutionize many industries, including security, environmental protection, agriculture, and industrial projects. However, the energy limitations of WSNs can pose a significant challenge, particularly in remote or inaccessible environments. To address this challenge, researchers have developed novel techniques and methods to extend the lifetime of these networks, including energy harvesting techniques, duty cycling, and power management.

This thesis focuses on utilizing wireless sensor networks powered solely by the environment, developing distributed inference techniques, and presenting real-world demonstrations of the proposed techniques in outdoor and indoor settings. Specifically, the thesis includes an outdoor demo using loopy belief propagation and two indoor demos using average consensus, to calculate the average temperature in a closed space and measure and estimate soil moisture on an agricultural field. Through this research, we aim to provide insights into the potential of WSNs and their applications, while also addressing the challenges associated with their deployment and management.

1.2 Ambiently Powered Distributed Network

There are two categories of networks: centralized and distributed. The focus of this thesis is on complex environments with limited energy supply, and demonstrates the potential for applications and networks that are not reliant on cloud computing, which can consume a lot of power and compromise data privacy. Instead, the projects in this thesis utilize pseudo-distributed networks, where operations such as measurements, estimations, and inference are performed in a distributed manner but are coordinated by a master computer, such as a microcontroller.

This work proves that it is possible to build a low-cost and low-energy distributed network that runs message passing and inference algorithms,

such as loopy belief propagation or average consensus, using only low-cost microcontrollers, RF antennas, RF transistors, and sensors. To meet the necessary energy and robustness specifications, the network is designed to be ambiently powered, battery-less (for the sensors), energy efficient in terms of data collection, and robust in terms of communication and sensor reliability. Furthermore, the vision for this work is to implement three offline applications, two interior and one exterior, that can make decisions without the support of the cloud.

1.3 Backscatter Radio

For monitoring an agricultural field in an outdoor environment, energy consumption for collecting measurements and estimating unknown values is a significant factor to consider. To extend the operating time of the network and reduce overall costs, the use of backscatter radios [1] was introduced. This solution involves using a smaller number of standard-cost boards along with additional low-cost boards.

Implementing a backscatter radio requires three components: a sensor tag, an unmodulated carrier transmitter (illuminator), and an embedded receiver [2], [3]. This approach allows for a reduction in the scale of cost and overall energy necessary to run the network, providing a more efficient and cost-effective solution. By leveraging backscatter technology, the network can operate with lower energy consumption, while still providing accurate and reliable data.

1.4 Outline

This project utilizes low-cost development kits and inexpensive scatter boards in order to implement a distributed inference wireless sensor network (WSN). It also investigates the feasibility of utilizing ambient energy sources to power the WSN (Chapter 2) and presents various software tools and configurations used for the development of the applications of this thesis (Chapter 3). Chapter 4 provides inference algorithms for the implementation of these applications. Furthermore, this project serves as a proof of concept that an ambiently powered WSN, that processes data in a decentralized manner using probabilistic, message-passing techniques (Chapter 5), can be implemented successfully. The work concludes in Chapter 6, which outlines future plans for the optimisation and furthering of this work.

1.5 Related work

This project integrates in-network message passing algorithms with energy harvested from ambient sources. While there are several existing works utilizing wireless sensor networks (WSNs), e.g., for agricultural field monitoring [4], many of them rely on costly and environmentally unfriendly edge

or cloud computing. Additionally, while there are other works that utilize inference algorithms for data processing, most of them are not in-network and require a central device for decision-making within the network.

Embedded Hardware

2.1 Embedded Hardware

Nowadays, it is possible to build a low-cost and low-energy network without handling low-level communication processes, such as modifying register states for radio communication. The boards used in this work, utilize various protocols and frameworks, with all the applications being developed in a higher layer of abstraction. Although the development of these boards requires a steep learning curve, they provide a plethora of examples that allow users to build, combine and customize their own applications. In this work, the Silicon Labs Thunderboard Sense 2 boards [5] are utilized as nodes in a sensor network. These boards are known for their low-power, low-cost and reliable nature and they communicate with each other through their radio modules. A node is considered connected to another node if it is within its communication range and can transmit or receive data to/from it.

2.1.1 Specifications

Thunderboard Sense 2 is a development platform that includes a broad range of embedded sensors and a very powerful radio.

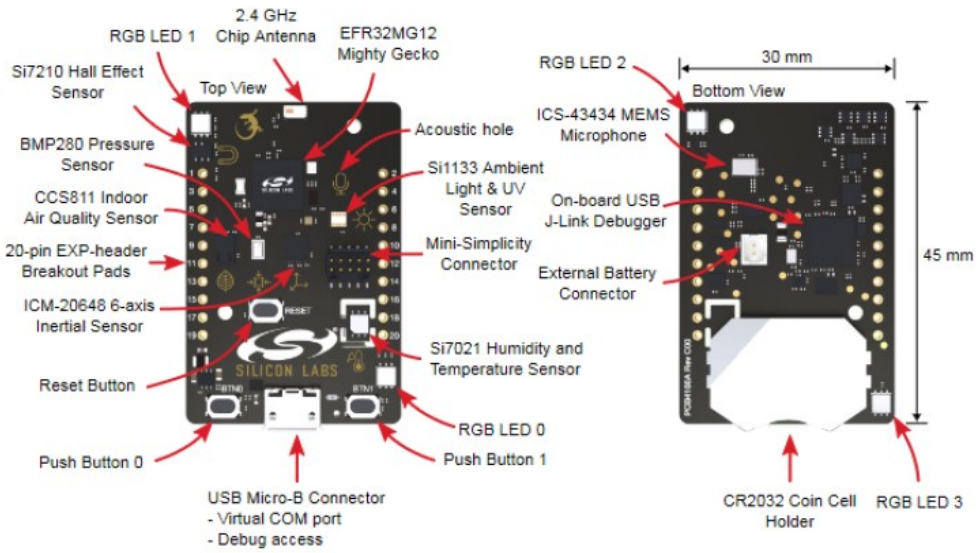

FIGURE 2.1: Thunderboard Sense 2 Hardware Layout [5].

The user interface includes:

- two push buttons and a reset button,
- one two-color LED,
- four high brightness RGB LEDs

Wireless Gecko EFR32™ provides:

- Wireless SoC EFR32MG12 Mighty Gecko with multi-protocol radio,
- ARM Cortex® M4 core with 256kB RAM and 1024kB Flash,
- Low Energy Consumption and Energy Modes,
- Flexible MCU peripheral interfaces,

The kit features are the following:

- 2.4 GHz radio configuration with on-board antenna,
- Segger J-Link Integrated Debugger,
- USB serial port,
- an 8Mbit serial flash nonvolatile memory,
- 20 breakout pads (SPI, UART, I2C and GPIO)

Thunderboard's embedded sensors are:

- Relative Humidity and Temperature Sensor Si7021,
- UV and Ambient Light Sensor Si1133,
- Pressure Sensor BMP280,
- Indoor Air Quality and Gas Sensor CCS811,
- 6-axis Inertial Sensor ICM-20648,
- Digital Microphone ICS-43434,
- High brightness LEDs,
- Hall-effect Sensor Si7210

The Thunderboards can be powered by: a USB cable with voltage of 5V that is regulated to 3.3V using a low-dropout regulator, or by a CR2032 coin cell holder or an external battery connected to the coin cell holder. To program and flash the boards, developers can use the Simplicity Studio software tool provided by Silicon Labs.

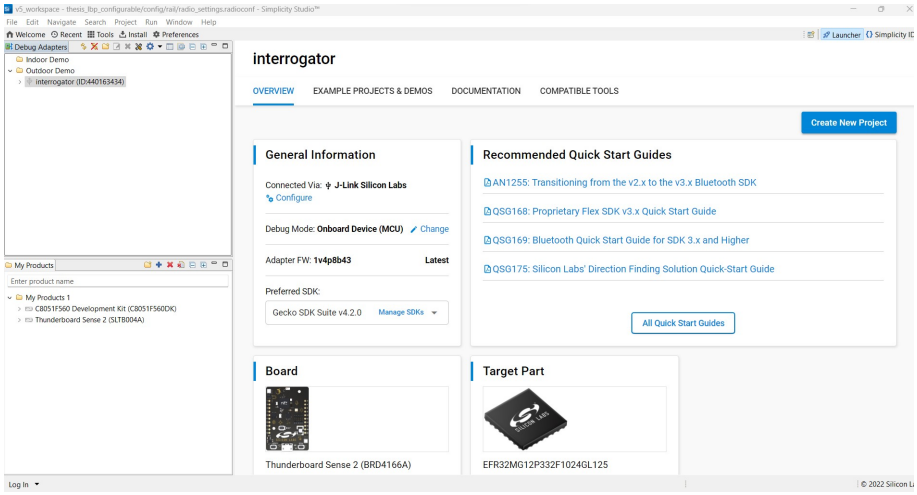

FIGURE 2.2: Simplicity Studio [6].

This software suite offers several features, including a radio configuration, a network analyzer, and an energy profiler. It also provides software examples for various protocols and frameworks, including Bluetooth, Platform, Proprietary, Thread, and Zigbee. For beginners, Simplicity Studio provides all the necessary tools and base examples to start programming hardware kits. To program the Thunderboards, developers must build the program, flash it, connect it to a computer via a USB Micro-B cable, and open a serial connection for debugging.

The Thunderboard stands out from other options due to its low power consumption, 32-bit architecture, multi-protocol capability, and low cost of only 20$.

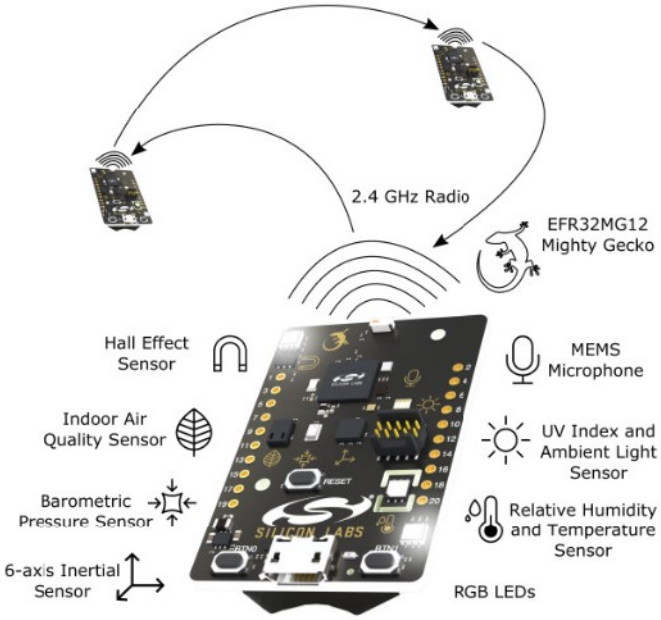

FIGURE 2.3: Thunderboard Sense 2 [5].

Temperature/Humidity Sensor

This development kit includes the Si7021 humidity sensor. The Si7021 is a digital relative humidity and temperature sensor that uses a capacitive humidity sensing element and a bandgap temperature sensor. It provides accurate, linearized sensor readings over a wide range of humidity and temperature values.

The Si7021 communicates with the microcontroller on the Thunderboard Sense 2 via an I2C interface, which allows for easy integration with the development kit.

This sensor measures temperature, with an accuracy of ± 1.0 °C over a range of -10°C to 85°C. This could be corrected using 2 point calibration [5]. But because in this project a denoising technique is used, the need for calibration became obsolete. This makes it a useful sensor for monitoring both humidity and temperature in a variety of applications, such as environmental monitoring, HVAC systems, and industrial automation.

The Si7021 has low power consumption, which makes it suitable for battery-powered devices and it also has a fast response time, which enables real-time monitoring of changes in humidity and temperature.

It's worth noting that accurate temperature measurements are critical in many applications, including environmental monitoring, industrial control, and medical equipment. As such, calibration is an important step in ensuring reliable and accurate measurements. When calibrating sensors, it's important to follow proper procedures, use appropriate reference standards, and account for any sources of error. In some cases, additional testing or analysis may be needed to identify and correct the sources of error.

Power Management

Modern microcontrollers offer energy modes to control the power they consume. These energy modes enable the microcontrollers to enter a low-power state and conserve energy, when the device is idle or not in use. There are typically several energy modes available, with each mode providing a progressively lower level of power consumption. This microcontroller offers five energy modes [7], [8] to manage power consumption: EM0, EM1, EM2, EM3, and EM4.

- Energy Mode 0 — Active/Run Mode

In this mode (EM0), both the CPU and the peripherals, are fully operational. This mode, although the chip is low powered, needs to be as short-timed as possible, because in a limited power environment, the batteries will be depleted very quickly. The current consumption in EM0 is 30mA.

- Energy Mode 1 — Sleep Mode

During this mode (EM1), the CPU is disabled, but all the peripherals are functional (IO). The system can stay in EM1 for a long time and wake up when needed from an external interrupt (radio). This mode has a current consumption of 11 mA.

•Energy Mode 2 — Deep Sleep Mode

In Energy Mode 2 (EM2), the high frequency oscillator is turned off. However, a 32 kHz oscillator and the real time clock are available for the low energy peripherals. The serial communication with the controllers is not available in this mode because it uses a high frequency clock. The microcontrollers need to be in this mode for the majority of time in order to maintain their available energy supply. This mode consumes only 2 µA.

•Energy Mode 3 — Stop Mode

In EM3 the low-frequency oscillator is disabled, while the low-leakage RAM ensures full data retention. The low power analog comparator or asynchronous external interrupts can wake-up the device. This mode consumes only 0.6 µA.

•Energy Mode 4 — Shutoff Mode

The EM4 is only suitable for applications where the use of a real time clock or RAM retention is not necessary. The board can wake up only with a reset or an external interrupt. The power consumption of this mode is 20 nA.

The power manager's software component controls the energy modes of the system, with the exception of EM4, which is not supported. In the main program, the user can specify the lowest energy mode that will not cause any issues in the application layer. The power manager then automatically selects the lowest possible energy mode based on this setting. Additionally, the power manager also offers a notification mechanism (subscribe) through which any software module can be notified of energy mode transitions through callback functions.

2.2 Scatter Tag

The scatter tag [3] is an ultra-low power and ultra-low cost board, which is responsible for acquiring the moisture readings from the sensor and sending them to a receiver. They are constructed in a modular architecture comprised of the scatter board (communication, sensor reading), the power board and the sensor board.

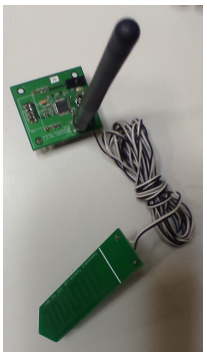

FIGURE 2.4: Scatter Tag [3].

FIGURE 2.5: Scatter Board [3].

2.2.1 Scatter Board

The board is comprised of an 8051 MCU and an RF transistor. Communication is achieved by turning the transistor (connected in the RF antenna) on and off. The sensor board is connected with the pins of the MCU. The clocking source for this particular MCU, is an external crystal oscillator, selected for stability reasons. These tags [2], [3] were constructed in the FABLAB of the TUC.

Power Management

- Active Mode

The MCU is active and has the highest power consumption. In this mode, using the 25 MHz external oscillator, the tag consumes 120 mA for the time needed to read one sensor and transmit the sensed value. The ADC polls the sensors connected to the MCU's pins and gathers the data.

- Idle Mode

While in this mode, the MCU idles and all the internal registers and memory retain their data. An interrupt or a reset can wake up the MCU. This mode consumes 4 mA.

- Sleep Mode

In this mode the real time clock (using the low-power internal oscillator) remains enabled, which is responsible for waking up the tag. In this mode, the tag consumes less than 1 µA.

After finishing with the process of sensing and transmission in active mode, the tags switch to sleep mode.

FIGURE 2.6: 3V 50F Capacitor.

2.2.2 Power Board

An ultracapacitor is a type of energy storage device that can store and release energy very quickly. It differs from a traditional battery, having a very high power density, but a lower energy density. An energy harvester is a device that can collect energy from the environment, such as solar energy or ambient vibrations, and convert it into electrical energy. By connecting an ultracapacitor to an energy harvester, we can store this energy and use it to power a microcontroller, without the need for a traditional battery.

By using this approach, we can develop batteryless devices that are more environmentally friendly and have a longer lifetime, since ultracapacitors can handle many more charge/discharge cycles than traditional batteries. However, it's important to note that this approach has limitations, as the energy available from an energy harvester can be limited and the power draw of the microcontroller must be carefully managed to avoid draining the ultracapacitor too quickly.

For this specific project [9] ruggedized electrical double layer energy storage capacitors up to 3 V operating voltage and with capacity of 50F, were used. There are shown in the below figure:

2.2.3 Sensor Board

The total amount of water, including the water vapor, in an unsaturated soil is called soil moisture [10] or soil water. It represents the water in land

surfaces that is not in rivers, lakes, or groundwater, but instead resides in the pores of the soil.

A capacitive soil moisture sensor is used to measure volumetric water content (VWC) in the soil. The sensor is a capacitor, made from two metal electrodes with a porous dielectric material between them. The larger the VWC, the larger the capacitance of the sensor will be (due to the change to the sensors dielectric constant).

Thus to measure VWC, measuring the sensors capacitance suffices. This is achieved by measuring the time ($t = ln(2)RC$) required for charging the capacitive sensor through a resistor of known value. The voltage of this capacitive sensor during charging is given by: $V_c(t) = V_s(1 - e^{\frac{-t}{RC}})$. The microcontroller of the sensor tag begins charging the capacitor and measures the time (t) required for it to reach a certain voltage, ($e.g.$, $\frac{V_{DD}}{2}$). Using the aforementioned equation, the capacitance of the capacitive sensor is then calculated by $C = \frac{t}{ln(2)R}$.

The measurement ranges between 100 and 300. But for practicality reasons this is normalised to 0 to 200.

FIGURE 2.7: Sensor board [11].

2.3 Solar Energy

2.3.1 Solar Harvester

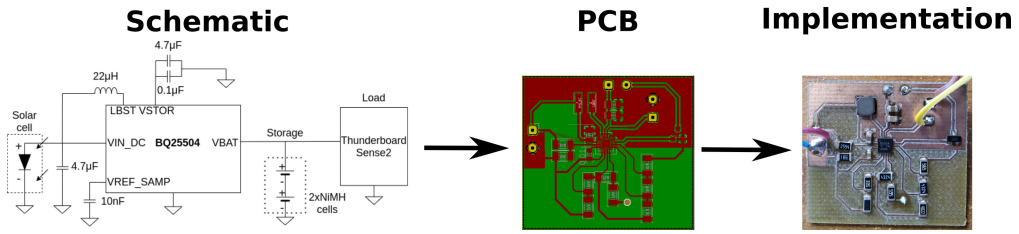

FIGURE 2.8: Solar harvester system from [12].

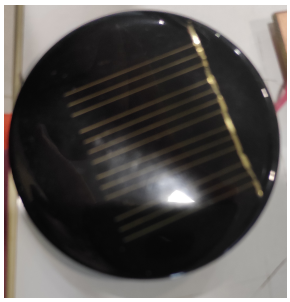

FIGURE 2.9: Solar Cell from [12].

In order to provide power to the Thunderboard Sense 2 board for the outdoor demo, we need to take into consideration the duty cycle (D) and the current consumption of the boards. The average current consumption is computed by the following formula:

$$I_{average} = I_{active} * D + I_{sleep} * (1 - D) \tag{2.1}$$

The solar cells [2] used for this demo have a circular area of $26.42cm^2$ and under full solar irradiance, the maximum power point (MPP) is in the elbow (knee) of the curve (Imp,Vmp).

$$P_{MAX} = V_{MPP} * I_{MPP} = 1.92V * 80mA = 153.6mW \tag{2.2}$$

The batteries utilized, are the low-cost, NiMh, of voltage 1.2V and of capacity Cbat = 100 mAh. To power the MCU 2-3.3V are needed, so two cells of total 2.4V connected in series, are enough to power them. We assume that, due to the age of the batteries at the time of the experiments, their efficiency is dropped to 60]]]]]]]]]]%. Their estimated capacity is 60 mAh.

In the application the total runtime of the measurements and the estimations is 3600 seconds or 1 hour. So in a day the duty cycle becomes 4.16%. The $I_{average}$ is 1.25 mA.

FIGURE 2.10: NiMh 2.4V 100mAh Batteries [12].

The charging time for the batteries, with no connected load, in specific conditions (full solar irradiance), is measured at 4 hours.

$$t_{chargenoload} = \frac{C_{bat}}{I_{charge}} hours$$

The charging time for the batteries, with a load connected, with the same conditions, is measured 4.10 hours.

$$t_{chargeload} = \frac{C_{bat}}{I_{charge}(1-D)} = \frac{4}{1-D} hours$$

The battery life is computed by:

$$batterylife = \frac{C_{bat}(mAh) * efficiency(\%)}{I_{average}(mA)}.$$

So for the old batteries the battery life would be around 48 hours with no solar radiation, but if we used new batteries (estimated efficiency 90% [13]) this would become 72 hours.

2.3.2 Solar Irradiance

Using regression equations, that compute the monthly average estimate of solar irradiance [14] for the climate of Greece, it was concluded that the probability of a power outage is small. During the winter months, when irradiance levels are at their lowest, there are at least 2-4 hours of sunlight per day. In this case, the absorbed irradiance is greater than $360\frac{W}{m^2}$, which constitutes the threshold when solar harvester start to charge the batteries. Therefore in the worst case scenario, i.e. January when the average solar irradiance is 225 $\frac{W}{m^2}$, in which the batteries will charge for at least 2 hours, they will still be able to run for 24 hours before being depleted.

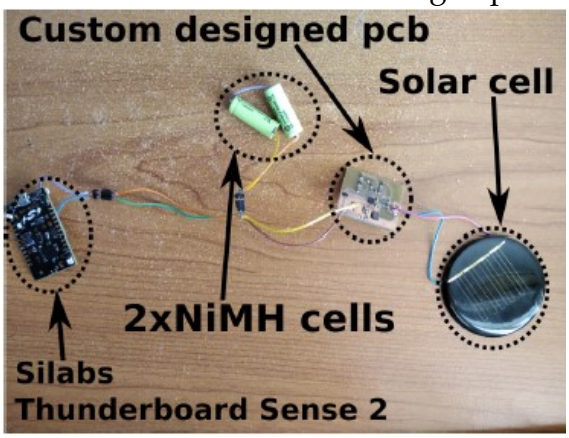

FIGURE 2.11: Solar harvester system from [2].

2.4 Peltier

A Peltier device is a type of solid-state heat pump that transfers heat from one side of the device to the other using electrical energy. It is also called a Peltier heat pump, solid-state refrigerator, or thermoelectric cooler. It can be used for both heating and cooling, but is primarily used for cooling. It can also be used as a temperature controller. Additionally, it can function as a thermoelectric generator, where a voltage is produced when one side of the device is heated, and a difference in temperature is created between the two sides (the Seebeck effect).

In this project the peltier devices [15] are supposed to be used as generators and will be connected to an energy harvester, in order to supply energy to the indoor demo (distributed variation). The power supply of the boards, in this variation, is to be studied in future work, as the time available for the completion of this thesis was insufficient.

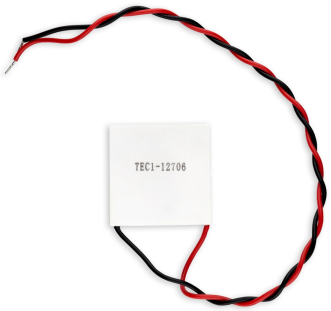

FIGURE 2.12: Peltier Element TEC1-12706.

Embedded Networking

3.1 Networking

3.1.1 Radio configuration

A proprietary protocol is the basis of this network and it was constructed using the Flex RAIL SDK [16]. For the communication of the microcontrollers, a frequency domain implementation was used, meaning that each terminal listens at its own channel (its board id) and transmits messages to a different channel.

Binary frequency shift keying (FSK) [17] is a simple and reliable modulation technique that is commonly used in applications such as digital communications, radio broadcasting and remote control systems. It is a form of digital modulation that is resistant to noise and interference, making it a popular choice for wireless communication. It involves using two different frequencies to represent binary data and was selected for this project. In FSK, a binary 0 is represented by one frequency, while a binary 1 is represented by another frequency.

FSK produces high level spurs (in integral multiples of the symbol rate). These spurious contents can be reduced by applying a Gaussian filter to the symbols. This filter smoothens the signal and removes these spurs, with the trade-off of reduced receiver sensitivity.

To transmit a binary signal using FSK, the binary data is first converted into a sequence of tones. The carrier signal is then modulated with these tones. In binary FSK, the carrier signal alternates between two different frequencies, depending on whether the input signal is a binary 0 or 1.

The frequency band utilized is the 2.4 GHz band, starting from the frequency of 2401MHz and ending on 2480MHz. There were multiple unlicensed frequency bands [18] that could be chosen, but a 2.4-GHz system has a longer range (if unobstructed) and generally requires a small antenna. This means that it is possible to construct a network only using the embedded antennas.

Some crucial information regarding the receiver specifications can be inferred, using the radio configurations of the examples provided by Simplicity Studio.

Some crucial information regarding the receiver specifications can be inferred, using the radio configurations of the examples provided by Simplicity Studio. The FSK receiver is a non coherent receiver. The minimum distance between the subcarriers $Df = \frac{1}{T}$ [19] is 1.2 kHz (deviation of 0.6 kHz) as the bit rate is 2.4 kbits/second. In order to maintain compatibility with existing backscatter radio tags the distance is set to 21.6 kHz (deviation of 10.8 kHz).

3.1.2 Network Layer

A number of functions that wrap the functions of RAIL API [20], were created, that help simplify the code in the application layer. These functions include the printing of the received/transmitted packet in the console, the setting up of the FIFO, the preparation of the package for transmitting and loading it in the FIFO, the unpacking of a received packet and the handling of a received packet.

3.1.3 Baton Protocol

The Baton is similar to the token used in the token ring protocol [21], in which one or more tokens is passed form host to host in a star topology. In this protocol only a host that holds a token can send data and tokens are released when receipt of the data is confirmed.

The aim of the baton protocol is to avoid conflicts and ensure that only one board will transmit information at any time in a network of boards. This is necessary because each board operates on different frequency channels and cannot both receive and transmit information at the same time. The baton protocol starts from the master node and is passed from one board to the next in a sequential manner. If a board is supposed to receive the baton but is detected as dead, it is skipped and the baton is passed to the next board instead. This implementation helps to ensure that there are no conflicts in the network and that information is transmitted efficiently and effectively.

3.1.4 MAC Address Protocol

Utilizing the MAC Address protocol, instead of the baton protocol for the implementation of the projects, would resolve network synchronization and energy supply challenges.

3.1.5 Backscatter Radio

The principle of backscatter [2], is that the illuminator streams an unmodulated carrier wave of frequency F_c . The antenna of the tag is impinged with this carrier and it reflects the induced signal.

An RF switch that alternates the antenna termination load between two values (Z1 and Z2), is connected to the antenna of the tag with switching frequency F_{sw1} and F_{sw2}. Using a spectrum analyzer we can see four subcarriers

appearing in the received frequency spectrum: one at F_c- F_{sw1}, one at F_c + F_{sw1}, one at F_c- F_{sw2} and one at F_c + F_{sw2}. The receiver is constructed in a way to receive the two subcarriers placed to the right of the carrier wave (F_c + F_{sw1} , F_c + F_{sw2}). These two subcariers represent the binary component of the 2-FSK. The time during which each subcarrier is present, is the bit duration (T_{bit}).

The receiver is tuned at 2445 MHz and configured to receive FSK modulated signals with a bit rate of 2.4 kbps. The distance between the FSK subcarriers is 21.6 kHz (deviation= 10.8). The illuminator is set to transmit the carrier wave at F_c=2444.89 MHz [2].

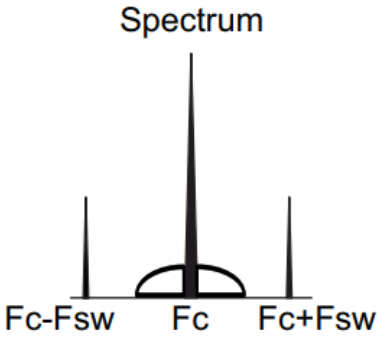

FIGURE 3.1: FSK scatter radio principle from [1].

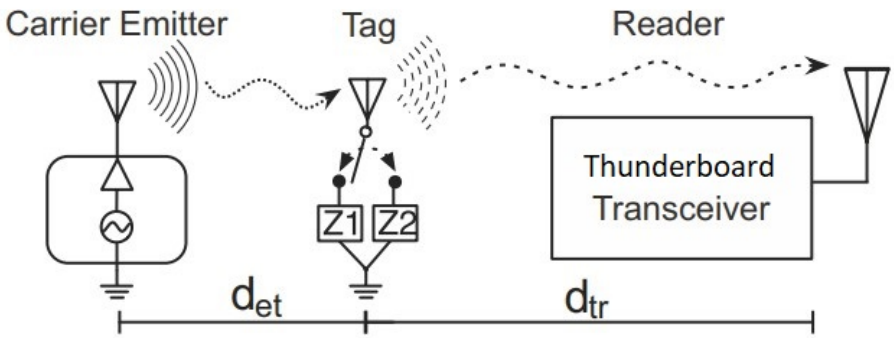

FIGURE 3.2: FSK scatter radio topology [1].

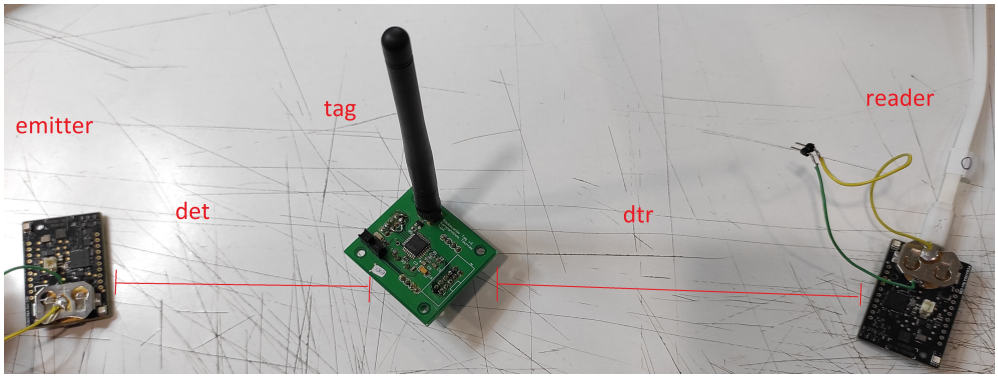

FIGURE 3.3: Backscatter radio topology.

3.2 Non Volatile Memory (NVM)

The microcontroller used in this project utilizes the Non-Volatile Memory (NVM) [22] to store data that needs to be preserved even when the energy mode changes (such as going into EM2) or when the application is flashed. The Gecko SDK v4.1.3 [23] provides a software component to manage the NVM. A wrapper class was developed for reading, writing and deleting data stored in specific memory locations (keys). The NVM provides a reliable way to store important data, as the contents will remain in the memory even after the energy mode changes or the application is updated. The user can only remove the data by giving a specific command.

3.3 Real Time Clock and Sleeptimer

The Sleeptimer [**Reference15**] is responsible for controlling the sleep and wake cycles of the boards during idle periods. The boards in the network go into deep sleep in order to conserve power and reduce energy consumption. During deep sleep, the high frequency clock doesn't work and the boards are only running on the low frequency clock. The Sleeptimer utilizes the low frequency clock and sends an interrupt to the board during its deep sleep to wake it up and execute the scheduled operation. In this way, the Sleeptimer helps manage the power consumption of the boards while still allowing them to perform necessary tasks during their idle period.

The Real-Time Clock (RTC) [24] provides timekeeping functionality. It can keep track of the current time and date. The RTC can generate periodic interrupts, allowing it to wake up the system at specific times, and can be used in a variety of applications, such as time-stamping events, scheduling periodic tasks, or maintaining an accurate time-of-day. The RTC can be used with the Sleep Timer to ensure that the system is awakened from deep sleep at specific intervals, without consuming excessive power.

3.4 Stack

In embedded systems, a finite state machine (FSM) is a widely used method for managing system states and transitions. An FSM is essentially a mathematical model that defines a system's behavior, where the system can be in a finite number of states and each state has a set of possible events that can cause a transition to another state. This makes the design of complex systems easier, as it breaks them down into smaller, more manageable pieces.

The implementation of a stack can greatly aid in the development of an FSM. A stack is a data structure that is used to store elements in a "last-in-first-out" (LIFO) order. This means that the most recently added element is the first one to be removed. This makes it a suitable choice for managing the state transitions in an FSM.

In the context of the text provided, the development of a stack for the project's FSM means that the different states can be pushed onto the stack, and when it's time for a transition to a new state, the current state can be popped off the stack, and the new state can be pushed onto it. This allows for a more organized and efficient management of the different states and transitions.

However, it's worth noting that the effectiveness of the stack implementation depends on the specific requirements of the project and the complexity of the FSM. In some cases, a simple switch-case statement may suffice for the FSM implementation. Nonetheless, the use of a stack can be a useful tool for managing the state transitions in many embedded system projects.

Wireless Sensor Networks as Inference Platforms

This thesis is a proof of concept that inference algorithms can be used in an environment where the power is limited. The algorithms that were used are presented bellow.

4.1 Algorithm 1: Loopy Belief Propagation

4.1.1 Loopy Belief Propagation

Loopy belief propagation (LBP) [25] is a tool in probabilistic graphical models. It is a method for estimating the marginal probabilities of unknown variables in a graphical model, which is a type of probabilistic model that represents the relationships between variables through a graph $\mathcal{E}$. In a pairwise markov random field (MRF), variables are represented by vertices in the graph and their relationships are represented by edges between the vertices.

LBP [26] operates by passing messages between the vertices in a graph, updating their belief values based on the relationships represented by the edges. This process is iterative and can be done in parallel, making it suitable for large, complex models. The algorithm is called "loopy" because it allows for cycles in the graph, meaning that messages can be passed between nodes multiple times. This allows for a more accurate representation of the relationships between variables, even in the presence of uncertainty or missing data.

Lets assume the variables x_i and x_j and their node potential $\phi_i\ (x_i)$, $\phi_j\ (x_j)$, representing the initial partial observations. The node potential conveys the unnormalized probability of node i being in state x_i without considering the influences of other nodes. The variables x_i and x_j are connected through an edge potential/compatibility function $\psi_{(ij)}(x_i, x_j)$. This function represents a joint unnormalized probability of nodes i and j being in states x_i and x_j, respectively.

$$p_x(x) = \prod_{i \in \mathcal{V}} exp(\phi_i(x_i)) \prod_{(i,j) \in \mathcal{E}} exp(\psi_{(ij)}(x_i, x_j))$$

Using this algorithm, it is easy to compute the marginal distribution of a pairwise Markov random field. This is done by iteratively passing messages between each neighboring variable on this graph. A message $m_{(i \to j)}(x_j)$ is the opinion of node i about the probability of node j being in state x_j. More precisely, the message a node i sends to its neighbour j, $m_{(ij)}$, tells him the possible state values of x_j based on local evidence, messages from other neighbors and the compatibility of x_i and x_j. This transmission scheme assumes that any variable i has to compute the $m_{(ij)}$ based on the belief messages coming from its neighbors excluding variable j, which is the target of the updated messages.

$$m_{i \to j}^{t+1}(x_j) = \sum_{x_i \in X} exp(\phi_i(x_i)) exp(\psi_{(ij)}(x_i, x_j)) \prod_{k \in N_{(i)} \setminus j} m_{k \to i}^{t}(x_i),$$

where $N_{(i)} \setminus j$ denotes the neighbors of i except for j.

Using the messages sent to it by all its neighbors and the local evidence at the node, each node estimates their beliefs. A belief b_i is the approximate marginal probability of node i being in state x_i, $p_{x_i}(x_i)$, which is computed from the converged messages and then normalized.

$$b_i^t(x_i) = exp(\phi_i(x_i)) \prod_{k \in N_{(i)}} m_{k \to i}^{t}(x_i),$$

where $N_{(i)}$ denotes the neighbor values of i.

After initializing the messages $m_{i \to j}(x_j) = 1$ for all the the edges $\mathcal{E}$ $(i, j) \in \mathcal{G}$, the messages are iteratively applied and sent.

A loopy graph with the messages that are exchanged between the variables, is shown below .

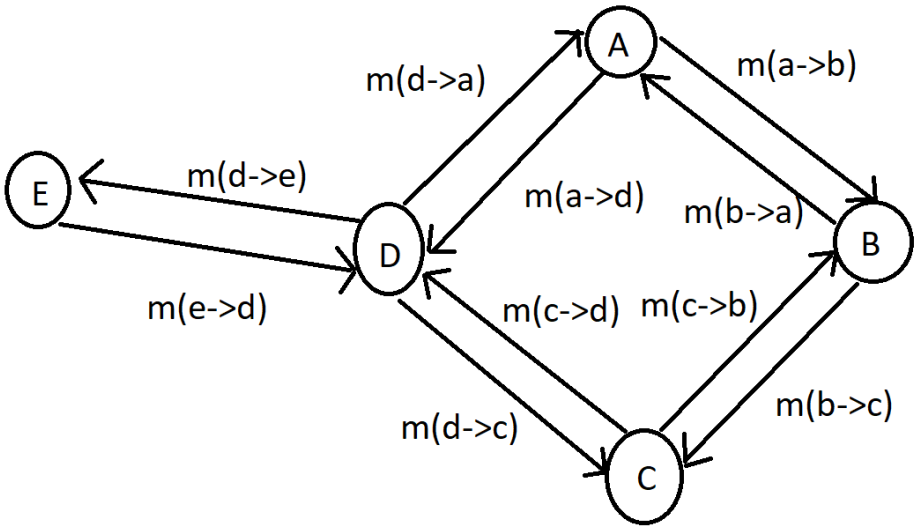

FIGURE 4.1: LBP Graph.

4.1.2 Iterative Proportional Fitting

Real-world examples include situations where samples in a specific area may be missing due to being unknown, unreliable or outdated. Iterative proportional fitting (IPF) [27], [28] can be used as a tool to estimate these missing values.

Iterative proportional fitting is a table weighting system, that ensures that its row and column totals' are in agreement with other sources. This is done through iterative calculations, resulting in a table of data, which is a joint probability distribution of maximum likelihood estimates. The iterations stop when the probabilities (these estimates) have converged within an acceptable limit.

Assume a table with rows containing some groups of objects and columns containing their possible status. Each element in this table represents the number of objects with a specific status. The table in figure 4.2 contains the measurements collected during the first year.

The row totals and column totals are the only available data provided for the second year (figure 4.3). Using IPF, the elements of the table are updated, to be consistent with column and row totals.

The ratio between individual items of the first and the second table, is proportional to the ratio of the row and column totals of the two tables.

Column1	Status1	Status2	Status 3	Row Totals
Group1	1306	83	0	**1389**
Group2	619	765	3	**1387**
Group3	263	1194	9	**1466**
Group4	173	1372	28	**1573**
Group5	171	1393	51	**1615**
Group6	159	1372	81	**1612**
Group7	208	1350	108	**1666**
Group8	1116	4100	2329	**7545**
Column Totals	4015	11629	2609	**18253**

FIGURE 4.2: First year table.

Column1	Status1	Status2	Status 3	Row Totals
Group1	1325.27	86.73	0	**1412**
Group2	615.56	783.39	3.05	**1402**
Group3	253.94	1187.18	8.88	**1450**
Group4	165.13	1348.55	27.32	**1541**
Group5	173.41	1454.71	52.87	**1680.99**
Group6	147.21	1308.12	76.67	**1532**
Group7	202.33	1352.28	107.4	**1662.01**
Group8	1105.15	4181.04	2357.81	**7644**
Column Totals	3988	11702	2634	**18324**

FIGURE 4.3: Second year table (IPF).

4.1.3 Edge Potential Functions Computation

In this project, the MRF model [29] is used to model the spatial correlations in environmental sensor networks. The MRF model is constructed using IPF on historical data and it serves as a representation of the relationships between variables in the network. The IPF algorithm is used, in this case, to adjust the parameters of the model with the purpose of matching them with the statistical distribution of the observed data. In order to avoid zero cell problems, the edge potential functions are initialized to one.

Once the MRF model has been constructed, the missing data in the network can be estimated using LBP. LBP estimates the probabilities of the variables in the network, given the observed data and the edge potentials in the MRF model. The algorithm works by iteratively updating the beliefs about the variables in the network, until they converge.

4.2 Algorithm 2: Average Consensus

Average consensus [12], [30] is an algorithm aiming to calculate the average value of a set of numbers in a distributed system. Each node in the system has its own value and wants to reach a mutual agreement on the average value of all the nodes. This algorithm has as advantages, that each node has knowledge about the estimated average and finally it can be used as a denoising tool to a network.

The basic idea of average consensus, is that each node in the network exchanges its value with its neighbors and updates its value based on the current average of its neighbors. This process is repeated iteratively until the values of all the nodes converge to the average value. The iterations stop, when the difference of the values of two consecutive iterations is lesser than a predefined threshold (the value has converged). The process of reaching an agreement on the average value is called consensus.

Average consensus can be used in various applications, such as networked control systems, sensor networks and distributed optimization. The advantage of using average consensus is that it enables nodes to reach a global agreement, without the need for a central coordinator.

Assuming a graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ are the graphs' vertices, with state x, and $\mathcal{E}$ the graphs' edges, defining the relationship between two vertices. Two agents are connected to this Graph via an edge, if they can exchange messages.

This algorithm determines the state x of each agent (vertex), which is the average value of all the states of all the agents in the network.

After some iterations, all agents' values converge to a fixed point x*:

$$x_i^* = \frac{1}{N} \sum_{J=1}^{N} x_j.$$

This convergence occurs according to the formula below:

$$x^{(l)} = \mathbf{W}x^{(l-1)},$$

where W is the weight matrix. This matrix is constructed using the degree d of the graph and the degree d_i of the vertex i.

$$\mathbf{W}_{ij} = \begin{cases} \frac{1}{d+1}, & i \neq j, \{i,j\} \in \mathcal{E}, \\ 1 - \frac{d_i}{d+1}, & i = j \\ 0, & i \neq j, \{i,j\}, \notin \mathcal{E}, \end{cases} \tag{4.1}$$

where d_i is the degree of vertex i and d is the maximum degree of $\mathcal{G}$.

Assuming a graph of 3 nodes, $d = 2$ and their initial values in figure 4.4.

$$x^{(0)} = [20, 30, 45]$$

The weight matrix of this graph is computed using the formula 4.1:

$$\mathbf{W}_{ij} = \begin{bmatrix} 2/3 & 1/3 & 0 \\ 1/3 & 1/3 & 1/3 \\ 0 & 1/3 & 2/3 \end{bmatrix} \tag{4.2}$$

In the final iteration, the nodes will converge to the average value of (20+30+45)/3 = 31.6.

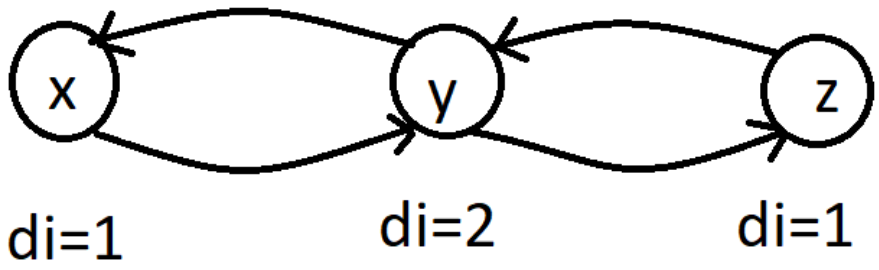

FIGURE 4.4: Average Consensus with 3 nodes.

Implementation of the WSNs

5.1 Distributed Measurement and Estimation System for Agriculture

In order to effectively model and monitor agricultural fields, frequent and accurate measurements are crucial. However, collecting detailed data from scattered sensors can be challenging, as it requires a lot of power and lacks robustness, making it difficult to maintain reliable readings. To address these challenges, a system called the Distributed Measurement and Estimation System for Agriculture (DMESA) has been proposed.

DMESA is a network of nodes, that are equipped with soil moisture sensors, placed in a grid to acquire measurements, estimate missing measurements, and provide them to the user. One key feature of DMESA is its ability to conserve energy and prolong the battery life of sensor nodes by putting them into a deep sleep mode. This energy-efficient mode helps maximize battery life and improve the system's overall performance.

Another critical component of DMESA is the use of two algorithms - LBP and IPF. These algorithms can estimate and predict data, even in the event of sensor node failures, which makes the system more robust. Using LBP and IPF, missing samples can be inferred from the alive nodes, ensuring that data collection and analysis remain accurate and reliable.

In summary, DMESA is a comprehensive solution for monitoring and modeling agricultural fields. It combines the ability to conserve energy with deep sleep mode and the LBP and IPF algorithms' capabilities of robust data processing and estimation, providing a reliable and effective way to gather and analyze data.

5.1.1 Sensor Measurement

In this work, two types of microcontrollers were utilized. The first one ("high" cost) is the Thunderboard Sense 2, which is responsible for wirelessly acquiring and storing measurements. The second one ("low" cost) the Sensor Tag, is responsible for acquiring soil moisture measurements from the ground and sending them to the receiver. So, in order to avoid confusion, in the future they will be referred to as: 'board' and 'sensor' respectively.

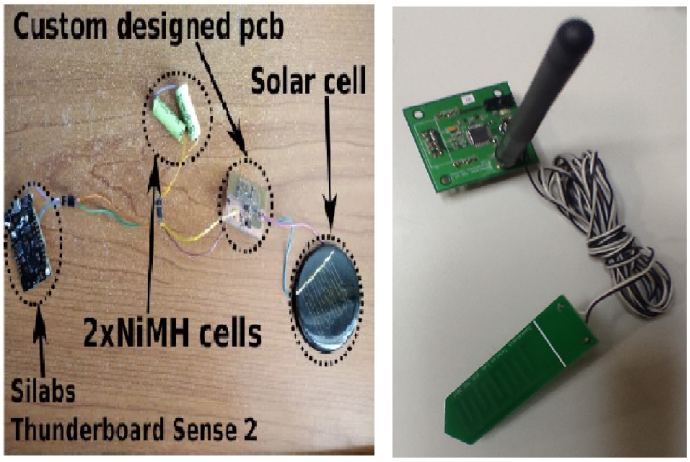

FIGURE 5.1: In the left picture is the Board and in the left is the Sensor.

The setup includes a network of boards and sensors. A number of sensors is "connected" to each board. This means that each board is responsible for the wireless collection of the measurements of these sensors. In the chapter 3 , the theory of backscatter radios is analyzed. In this network, during the measurement phase, one board acts as a carrier emitter (illuminator) and another board is responsible for the acquirement of the measurements (receiver). The scatter tag is positioned between them, either near the illuminator or near the receiver and is connected (wired) to the soil moisture sensor.

For an inference network, containing 6 boards and 12 sensor tags (two sensors assigned to each board) the illumination starts from the second board and the receiving (of the measurement of the first two sensor tags) starts for the first board. This happens until all the boards have illuminated their neighbors tags (if available) and the neighbours have acquired their measurements. The measurement process is done in a time schedule predefined by the user.

The user must define the sensors that will be assigned to each board, as well as their spatial relationship. The sensors are placed in an orthogonal grid and their relationship is defined by their cartesian coordinates. Two of these sensors (x_i, y_i), (x_j, y_j) are considered neighbours, if they are aligned (have one identical coordinate y_i=y_j or x_i=x_j). For example, the sensor in the grid placed in the position (0,0) and the sensor placed in the position (0,1) have the same x coordinates so they are neighbours. But the sensors placed in (1,0) and in (2,1) are not neighbours. The sensors assigned to each board and their grid layout are presented below:

- Board0 : (sensor0) , (sensor1)
- Board1 : (sensor2) , (sensor3)
- Board2 : (sensor4) , (sensor5)
- Board3 : (sensor6) , (sensor7)
- Board4 : (sensor8) , (sensor9)
- Board5 : (sensor10) , (sensor11)

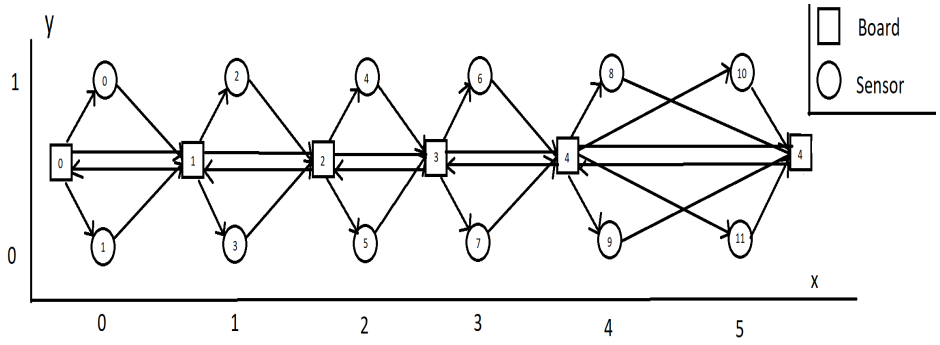

FIGURE 5.2: Graphic representation of the topology of the network. Boards are depicted as the squares and sensors are depicted as circles. The arrows from a board to a sensor indicate the direction of illumination. The arrows from a sensor to a board indicate the transmission of a measurement and from a board to another board internal communication.

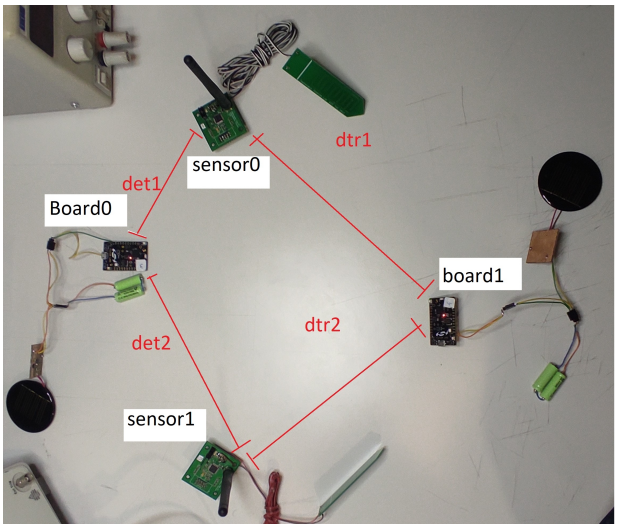

FIGURE 5.3: Image from a segment of this network.

The user is also responsible for defining the schedule in a way that doesn't affect the robustness of the network (battery consumption). It's decided that nine times per day soil moisture measurements and estimations will be computed by the network. A low duty cycle is chosen, in order to increase the network robustness. Only three of the soil moisture measurements are deemed critical (8:00:00,16:00:00,24:00:00) and are saved in the NVM consisting the measurement history available to the algorithm.

- 00:00:00 save in NVM estimate
- 02:00:00 estimate
- 04:00:00 estimate
- 08:00:00 save in NVM estimate
- 12:00:00 estimate
- 14:00:00 estimate
- 16:00:00 save in NVM estimate

- 20:00:00 estimate
- 22:00:00 estimate

5.1.2 Implementation

The setup for this network, includes a board, that receives the results of the measurement and estimation process, connected to the computer with a wire. This board, from now on, will be referred to as the interrogator board. The communication from the computer to the interrogator board is serial. Via the User interface, the user is able to set up the Current Local Time and request the network's most recent estimates.

For coordination and synchronization purposes, the network of boards has a master board. This board contributes normally to the measurement and estimation process. Additionally, the master is responsible for sending the Current Local Time to all the boards and ordering measurement collection from (all) the sensors. Generally, because the real time clock is not accurate, it needs to be synchronized multiple times per day. For that reason before the slaves go into deep sleep, they request the Current Local Time from the master node. For robustness purposes, it is necessary for this board to never go into deep sleep and to have an uninterruptible power supply, or the network will stop functioning.

There are two instances, in which the interrogator board has to be connected to the computer. The first, is when the user sets up the Current Local Time, a process necessary only once during the setup of the network. The second is when the user wants to request the most recent results of the estimation process from the network.

The steps necessary to setup the network are described below. First, the user flashes the devices with the DMESA (each one having a different board id), places them in the agricultural field, after connecting them with the solar harvesters. Second, the user places the sensors, in an orthogonal grid topology (with the soil moisture sensors probed into the ground). Third, they connect the interrogator board to the computer and choose to set the Current Local Time of the network using the UI. The computer program sends the Current Local Time to the interrogator, via UART and then the interrogator relays it wirelessly to the master node. Then, using the respective button embedded in the Thunderboard Sense2, the user resets all the boards except the master board. The slave boards request the Current Local Time from the master and the master transmits it. The slaves compute the time during which they will deep sleep, until the next estimation process (according to the estimation/measurement schedule) and go to sleep.

When its time for the boards to wake up and perform the estimation process, the master starts the neighbour discovery process. This means that all the slaves that are still alive send a heartbeat, so that the master knows and updates their status (alive/dead). The master sends the results to the alive slaves. All the boards adapt their topology graphs according to the status of their neighbouring boards.

If the demo was designed and implemented using the MAC Address Protocol, the boards alive would perform node discovery individually and then they would collectively decide on a consensus for the graph topology. They would also be able to reach a consensus for the Current Local Time without the need of a master node. This substitution would render the use of constant power supply for a node (master node) obsolete.

After neighbour discovery process has concluded, the sensor measurement phase begins. When all the boards have completed their illumination and have received their measurements, the IPF algorithm is called in order to compute the edge potential functions. This is done by receiving all the neighbouring boards' histories. Then, using these functions, the outgoing LBP messages are computed and sent to the neighbours (either internally or externally).

For example, we know that board0, which is responsible for sensor0 and sensor1, and the board 1 responsible for sensor2 and sensor3. Sensor0 and sensor1 are neighbours and their LBP messages will be exchanged internally in board0. On the other hand, although Sensor1 and Sensor2 are neighbours, their boards (board0 and board1) exchange their messages externally. The boards receive the messages during each iteration and in the final iteration they compute the beliefs of their sensors. Each slave board, sends their sensor measurement/estimation to the master board. All the slave boards go to deep sleep and the master sleeps (can receive messages from radio) until the interrogator requests the estimations. Then, the interrogator receives the measurements and estimations from the master board and sends them to the computer. The results of each sensor are shown in the constructed user interface after the user asks for it.

5.1.3 Timers

The boards provide a hardware Timer [**Reference23**]. When the timer expires, a callback function is called. This application requires multiple timers in order to function, so we utilise multiple software timers provided by RAIL, by enabling a software multiplexing layer.

One example where timers are necessary, is to restart an operation (measurement or LBP), in case of an unexpected error. Such an error is the case where the Baton is stuck. This means that a board died in the middle of an operation, so after a predefined time period, during which the operation is not over, the process starts all over again by first performing board discovery. There is also a timer that ensures that no slave board stays awake for more than 10 minutes (in case of an unexpexted error), thus avoiding totally depleting its energy.

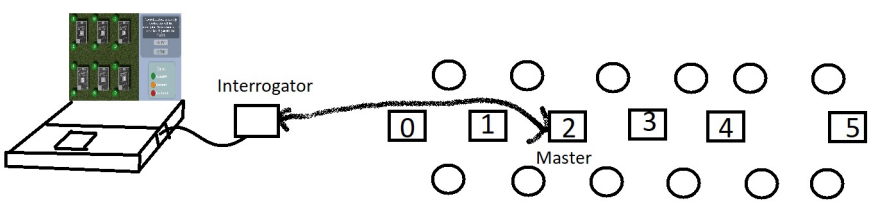

FIGURE 5.4: Setup.

5.1.4 Radio Configuration

Due to the fact that each board communicates with other boards and receives messages from the scatter tags, but is also responsible for the illumination of sensor tags, two different setups are stored in the boards using a protocol provided by RAIL called Channel-based Multi-PHY [**Reference25**].

To effectively achieve the Channel-based Multi-PHY configuration [31], channel groups must be created each of which has a different radio configuration. Changing between these channel groups is handled by RAIL API. When the program requests to receive or transmit from/to a specific channel then RAIL selects the configuration, in which the channel belongs to.

This project contains 2 radio configurations. One starting from channel 0 to 49, with base frequency 2445.00 Mhz and the other starting from channel 50 to 99 with base frequency 2444.89 Mhz. The first radio configuration is used for the communications of the boards and the tags to receive the sensor tags' measurements and the second is used for the illumination of the sensor tags.

These radio configurations use 2-GFSK modulation with 21.6 kHz deviation and maximum available transmission power of 10.4 dBm.

5.1.5 Packet Configuration

The packets comprised of the preamble, the sync word and the payload. The payload has a constant size of 16 bytes for the communication between the boards and 6 bytes for the measurement collection. Typically the 16 byte packet has the format of figure 5.5.

Depending on the current state of the state machine, the structure of the messages differs.

The packet for the measurement collection from the sensors is comprised of the message id (2 times for error detection), the destination board, the sensor's id and the 2 bytes of the sensor's measurement. The 6 byte packet has the following format of figure 5.6.

message id	source	destination	Data	Data	Data	Data	Data	Data	Data	Data	Data	Data	Data	Data	Data
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

FIGURE 5.5: Boards' packet.

message id	message id	destination	Sensor id	meas1	meas2
0	1	2	3	4	5

FIGURE 5.6: Sensors' packet.

5.1.6 Command Line Interface

The Command Line Interface (CLI) [32] contains all the functions of each cli command. The available commands enable the communication between the user and the boards . A brief overview of the functions, created for the purposes of this application, is provided below:

- "info": This command prints the unique id of this MCU.
- "help": This command prints all the available commands of the CLI.
- "nvmreset": This command completely wipes the non volatile memory, which contains the assigned to it, sensors' histories.
- "nvmdisplay": This command displays all the contents of the NVM.
- "estimate": This command, called exclusively by the interrogator board, is for requesting from the master board for the results of the sensors' measurements and estimations (IPF, LBP).
- "clock": This command sets the master board's real time clock. It expects input time conforming to the 24 hour format, starting from 00:00:00 and ending at 23:59:59.
- "time": This command prints the master board's current time on the console.

5.1.7 Initial Setup

Before the execution of the main function, that runs repeatedly, some initializations and instantiations are necessary, for the correct execution of the application. The first is to instantiate a handle for RAIL, which is used by the majority of the RAIL API functions. It is also important to set up a FIFO structure for the transmission mechanism and to initialize the Non Volatile Memory. Other components that require initialization include: the power manager, the sleep timer, the software enabled timer and some variables.

5.1.8 Configuration

This program is written so all the parameters concerning this project are configurable. For example, the user can configure the number of boards, the numbers of sensors for each board, select the master board, the number of LBP iterations and the time on all of the timers.

The number of sensors "connected" to each board is a trade-off with the accuracy of the soil moisture measurements. The table containing each value of measurement and its' frequency, for all the sensors, is saved in the non volatile memory. This memory has a finite size. So because the memory has 256 available places, this means that if the board is responsible for 2 sensors, then the range of the soil moisture sensor is downscaled from [1, 200] to [1, 100]. But if the board is responsible for 4 sensors then the range is downscaled from [1, 200] to [1, 50] taking up 200 places in the memory again. The read/write operations from/in the NVM are costly, both concerning their energy and the time consumption. This is due to the fact that memory access

is a linear process and when the information accessed is located near the end of the memory the traversal of the memory is necessary.

For the reasons mentioned above, in this test setup, even though it was possible to downsize the range from [1, 200] to [1, 100], we chose that there will be 25 distinct levels of soil moisture instead. Resulting in a range of [1, 25] and 2 sensors for each board. Therefore 50 total values will be saved in the NVM.

5.1.9 IPF Implementation

For this specific implementation, as mentioned above, there are 2 sensors assigned to each board and 25 distinct values of soil moisture. This means that there will be a 2D table M with its rows and columns (x and y coordinates) respectively corresponding to the soil moisture values of the first and second sensor assigned to the board. The table is stored in the NVM and each of its elements is a joint probability of the first sensor taking the value x_i and the second sensor taking the value y_j: $P_{X,Y}(X = x_i, Y = y_j) = M(x_i, y_j)$. So the purpose of IPF is to iteratevely adjust these probabilities, according to row and column totals.

5.1.10 User Interface

For a straightforward initialization and depiction of the sensor network, a user interface was created using the Unity 3D application development environment [33]. Unity is a very popular engine both for application development and prototyping. It was selected to create a demo of our application user interface, taking into consideration its many capabilities, developer friendly design environment, extensive toolkits and community support.

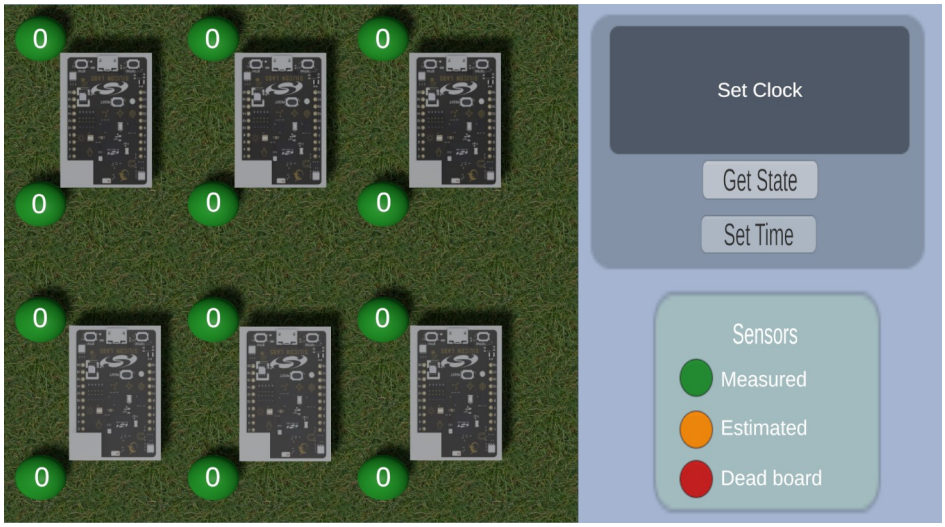

FIGURE 5.7: Applications' User Interface.

A User Interface for the communication with the Interrogator was built, using Unity and Ardity [34], which is a Unity component that permits the serial communication with a microcontroller, like Arduino. Using this application the user can set the local time of the network of boards and get the most

recent estimations. For each board and its sensors, there are three possible outcomes depicted in this UI.

These include :

• A Sensor's X estimation: Y, which has NOT been received.

This means that for some reason, in this round, the board has malfunctioned (no battery, or dead), so the network decides its value Y.

•B Sensor's X estimation: Y, which has been measured.

This means that the board measured the Y value for the X sensor.

• C Sensor's X estimation: Y, which has been estimated.

This means that the sensor X in this round of measurements, for an unknown reason, didn't send its measurement. For that reason, the network decided its value Y.

Note that the topology of the network, as presented in the user interface, does not correspond to the real topology of the constructed network for this experiment. The real topology is the one described in the section 5.1.1 and is depicted in **Figure 5.2**.

The created UI for the application is shown in the image above. On the left part of the screen, a basic representation of the grid is depicted. An instance of the orthogonal grid presented, containing 6 Thunderboard Sense2 boards. Each board has: a window on their lower left corner, indicating their status and another window in their center indicating their ID. The *alive* status is represented with *green* colour and the *dead* status is represented with *red* colour. It also contains 12 sensors, depicted as spheres with a number inside them, indicating their measurement or estimation. If the sphere is *green* the number inside corresponds to a *measurement*, if its *orange* then it corresponds to an *estimation*, else if it is *red* then that means that the sensor is *dead*.

In the upper right there is a screen indicating the status of the network and two buttons to set the Current Local Time and get the current state of estimations from the network. In the lower right of the screen there is table indicating what each sensor colour means.

As described in the subsection 5.1.2, the user can use the designed UI, to set up the Current Local Time, a process necessary only once during the setup of the network and to request the most recent results of the estimation process from the network. The following figures depict the user interface possible outputs for different user input actions in these two scenarios.

In the **Figure 5.8** the user has attempted setting the Current Local Time. The action is successful and a respective message appears on the screen informing the user on the state of the synchronization process.

In the **Figure 5.9** the user has attempted setting the Current Local Time. The action is unsuccessful and a respective message appears on the screen informing the user on the possible causes of the process failure.

In the **Figure 5.10** the user has pressed the Get Status Button. On the grid, users can see information about: the status of the boards and sensors and their estimations and measurements.

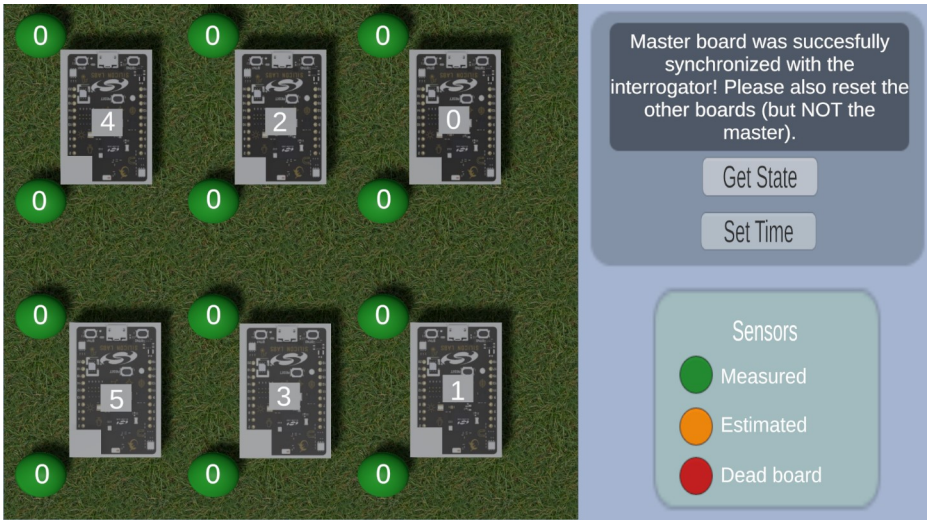

FIGURE 5.8: Instance of the application after setting the Current Local Time successfully.

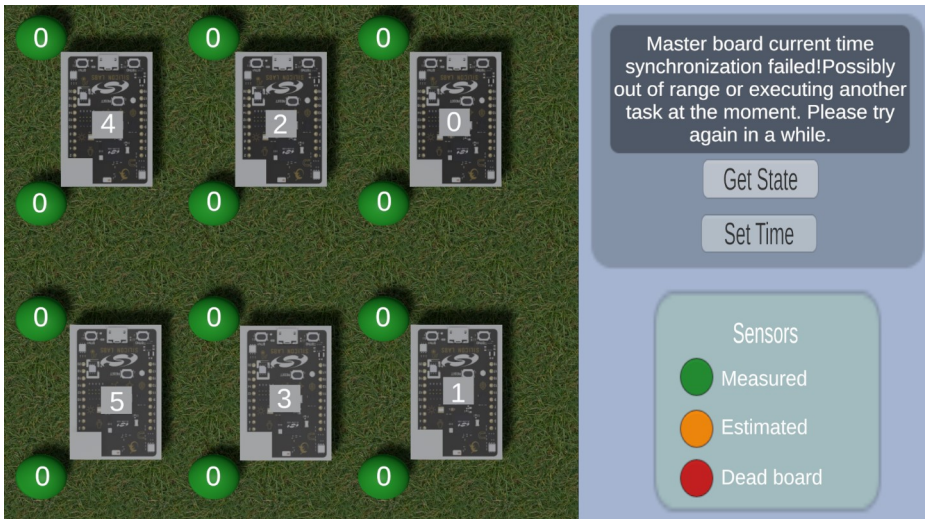

FIGURE 5.9: Instance of the application after trying to set the Current Local Time.

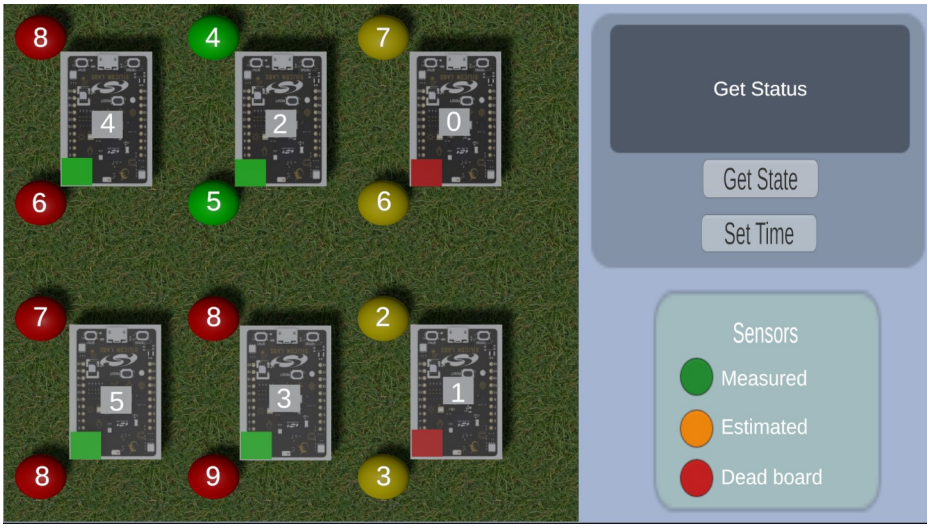

FIGURE 5.10: Instance of the application after running an estimation.

The current user interface in this project is designed to work with 6 boards and 12 sensors, but may not be scalable for larger networks. To create a user-friendly and scalable UI, additional research is necessary. This research should include an understanding of the use cases and personas for the system, as well as considerations for its scalability and flexibility. This can be achieved through user research, persona creation, and testing the interface with different configurations. It's also important to consider the underlying technology and infrastructure of the wireless sensor network, including hardware and software components, communication protocols, and data processing requirements. By taking a holistic and user-centered approach to the development of both the wireless sensor network and its user interface, we can create a system that is both effective and easy to use.

5.1.11 Results

The results of the first experiment, conducted in the gardens of ECE in the technical university of Crete, are promising. The following figure is a Google Maps satellite view of the area where the experiments were conducted. The satellite image was edited and red and yellow dots were added representing the network topology as shown in figure 5.2.

An estimation process ran once, while all the boards were alive and gathered measurements from 8/12 total sensors. This is a success rate of 66%. In this experiment the duty cycle of the sensors was 100% as they had as a power supply a CR2032 battery.

The figure 5.12 is a field view of the network, set up in the location indicated by the figure 5.11. One board is connected to a powerbank as the solar harvester broke right before the experiment. The image was captured during the first experiment run, after assembling all its components, outside the LAB:

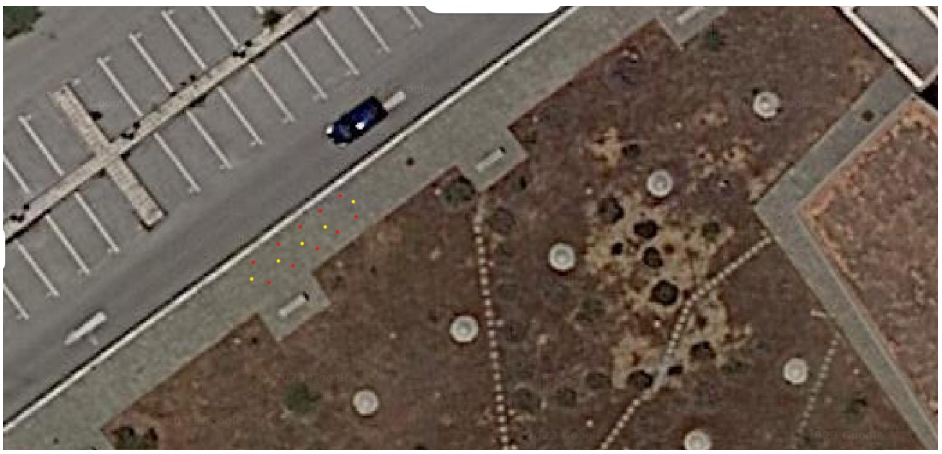

FIGURE 5.11: Grid in ECE garden.

FIGURE 5.12: Photo of the grid in ECE garden.

During the second experiment the Scatter Tags were connected to the ultra-capacitors and to the energy harvesters (figure 5.13). So their duty cycle was adjusted to approximately 20%. Both the boards and the sensors were fixed to a stand (figure 5.13, figure 5.14). In this experiment 11/12 sensors were available and during the first attempt the network received measurements from 4/12 of the sensors and in the second attempt the network received from different 4/12 sensors. When the network is operating in its equilibrium state, it is expected that not all sensors will send measurements at each scheduled operation.

FIGURE 5.13: Sensor Tags in the field.

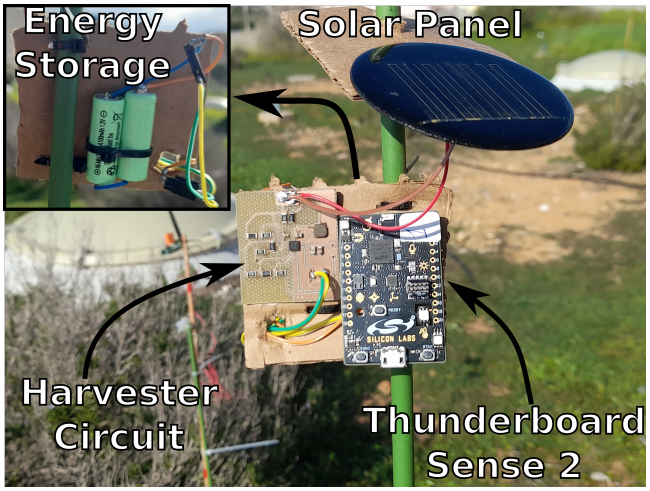

FIGURE 5.14: Boards in the field.

FIGURE 5.15: An instance of this network.

5.2 Distributed Averaging System for Handling Indoor Temperature

The importance of temperature control in homes and buildings that store perishable goods cannot be overstated. These goods have specific temperature requirements for preservation and even small deviations from the desired temperature range can lead to spoilage, loss of quality, or even health hazards.

Traditionally, a single thermometer was used to measure the temperature in a building. However, this method is prone to errors, as the temperature can vary significantly from one location to another, within the same building. Moreover, if the thermometer itself is faulty, the readings can be inaccurate, leading to incorrect temperature control.

To overcome these challenges, the concept of the Distributed Averaging System for Handling Indoor Temperature (DASHIT) was proposed. This system uses multiple temperature sensors, placed throughout the building, to measure the temperature. The readings from these sensors are then collected and processed, to determine the average temperature of the indoor environment, using the average consensus algorithm. By taking multiple measurements and calculating the average temperature in a decentralized manner, DASHIT provides a more accurate and reliable representation of the indoor temperature compared to traditional single thermometer systems. This system has 2 different variations, each one having its' benefits and its' shortcomings.

5.2.1 Centralized Variation

The aim of this project is to create a robust network, by utilizing the deep sleeping capabilities of micro-controllers. The network operates in a distributed manner, with a master node coordinating and synchronizing the process. This master board is responsible for sending the Current Local Time to all the boards and collecting measurements from all available boards in the network. It is essential for the master board to have an uninterrupted power supply, to ensure the network's reliability. The master board has a limited communication range and must be able to communicate with all boards in the network.

To begin the process, the user flashes the devices with DASHIT and places them in an indoor space connected with their peltier devices and batteries. The user places them according to his desired topology of the boards, as long as all the boards can communicate with each other. The master node is then connected to the computer and the user sets the Current Local Time, using serial communication. After resetting all the slave nodes, they request the time from the master node and then enter deep sleep mode, until the next scheduled process.

When it's time to perform the algorithm, the boards start with neighbor discovery, where each slave sends a heartbeat to update the master board.

The master board keeps the slave boards informed of the network's status. According to this status, each board constructs the graph with all the alive boards that are going to contribute to this process. The temperature measurement process follows, where all nodes acquire measurements and exchange their state variables, until convergence. This process is repeated several times a day, based on a user-defined schedule. During times of inactivity, the boards go into deep sleep mode, in order to conserve energy. In case the unavailability of a board is detected by the network, during the average consensus operation, node discovery is repeated and the graph is reconstructed. If the demo was designed using the MAC Address Protocol, the boards would individually perform node discovery and then they would collectively decide on a consensus for the graph. They would also be able to reach a consensus for the Current Local Time without the need of the master board.

This network is designed to have low energy requirements and is supposed to be connected to an energy harvester and a peltier device. Due to limited time the network wasn't tested with this configuration.

Time schedule

- 00:00:00 average
  - 02:00:00 average
  - 04:00:00 average
  - 06:00:00 average
  - 08:00:00 average
  - 10:00:00 average
  - 12:00:00 average
  - 14:00:00 average
  - 16:00:00 average
  - 18:00:00 average
  - 20:00:00 average
  - 22:00:00 average

Timers

The software implemented timers of this microcontroller were used for the implementation of this project, in order to assert that no problem exists in the network. For example, there is a timer expiring after a predefined period of time, forcing the nodes to deep sleep, irregardless of the reason for the timer timeout. The cause of the timeout could either be an error or a running task timeout.

Radio Configuration

This project contains one radio configuration. It starts from channel 0 to 49, with base frequency 2445.00 Mhz and is used for the communication of the boards. This radio configuration uses 2-GFSK modulation with 21.6 kHz deviation and maximum available transmission power of 10.4 dBm.

Command Line Interface

The Command Line Interface (CLI) contains all the CLI command functions. These commands are provided below:

- "info": This command prints the unique id of this MCU.
- "help": This command prints all the available commands of the CLI.
- "average": This command, called exclusively by the master board, is for returning the result of the average consensus process.
- "clock": This command sets the master board's real time clock. It expects input time conforming to the 24 hour format, starting from 00:00:00 and ending at 23:59:59.
- "time": This command prints the master board's current time on the console.

Initial Setup

To ensure proper execution of the application, a number of initializations and instantiations must be performed prior to running the main function. The first step is to instantiate a handle for RAIL, as it is used by the majority of the RAIL API functions. Additionally, it is important to establish a FIFO structure for the transmission mechanism, which is essential for reliable data transfer. Other components that require initialization include: the power manager, the sleep-timer, and the software-enabled timer. Finally, some variables (i.e. the weight matrix or the transmission power) for the average consensus algorithm need to be initialised in order to enable accurate data processing and analysis. By performing these necessary initialisations, the application can run smoothly and without errors, ensuring optimal performance and functionality.

Results

In order to test the network, two topologies were implemented. The first one being a chain topology, with the boards having distance of 1 meter between them and the second being a circle network. The results were compared to a reference temperature and were deemed satisfactory. In addition to its intended function the algorithm was observed to be a sensor measurement denoising tool. Assuming that all the boards are alive, the connectivity graphs representing the two network topologies are presented below:

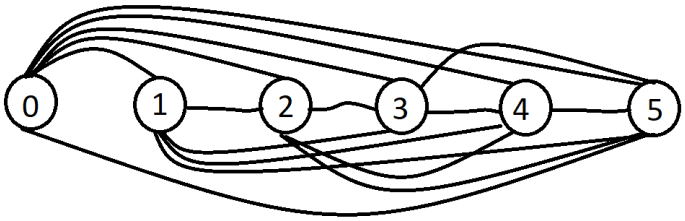

FIGURE 5.16: Chain Grid.

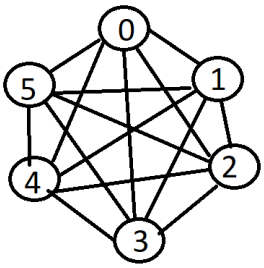

FIGURE 5.17: Circle Grid

5.2.2 Distributed Variation

This version of the network is designed to expand its range, with the requirement that every node must be able to communicate with at least one other node. The communication scheme is user defined. During normal operation, all nodes are powered by a constant power supply, using power mode EM1. The average consensus process only occurs after a user request, allowing for greater energy efficiency. Users can communicate (serially) with any board and request the network's average. The sleep timer is not used in this version. The radio configuration and Command Line Interface used, are the same as the Centralized Variation.

The primary goal of this version of the network is to create a more extensive and robust network, that allows greater network range. To achieve this, each node must be able to communicate with at least one other node, increasing the network's connectivity and reliability.

To set up the network, the user inputs the graph of nodes, specifying their connectivity and the route that the baton will take in order to pass from all the boards. This process ensures that the network is set up correctly and that all nodes are communicating with at least one other node.

The graph of this network is hard-coded to each board rendering the network useless in case a board dies. That's why it is important for the boards to have a constant power supply, in this variation. If all the nodes had a MAC address this would be resolved, as all the boards would have knowledge of the depth of the graph and their degree in order to adjust their weights and readjust the graph based on the new status of the network.

Overall, this version of the network is similar to the Centralized Variation, in terms of radio configuration and command line interface, but it focuses on expanding the network's capabilities by ensuring each node can communicate with at least one other node. It operates using a constant power supply and only initiates the average consensus process when a user requests it, making it an energy-efficient solution.

The test grid is presented below:

The path that the baton will take is: 3->2->1->0->5->4->5->1->2.

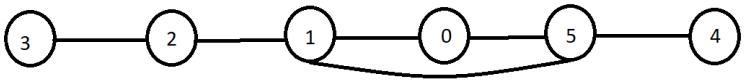

FIGURE 5.18: Distributed Average Consensus Grid.

Conclusions

6.1 Conclusion

The development of this project is significant, as it demonstrates the feasibility of utilizing low-cost and low-power networks, for in-network inference. With the incorporation of solar harvesters, the network can function for extended periods of time without interruption, making it highly advantageous in applications where power is limited or unavailable. The use of backscatter radios is also proven to be highly beneficial, as it allows restricting the cost of the overall network.

The outdoor demonstration can be used to measure and estimate soil moisture from different points in a field. Due to early design choices and unavailability of time, this network has some limitations. The first one being the need for a constant power supply for one board, as well as the limited range of the network. Also, there is a limit concerning the number of sensors assigned to each board due to the limited space available in the non volatile memory.

The indoor demonstrations can be used to measure the temperature of an enclosed space from multiple points. Due to early design choices and unavailability of time, though, these 2 network variations have some limitations. For the centralized version these limitations are the need for a constant power supply for the coordinator node (master) and the limited range of the network. For the distributed version the problems are the constant power supply of the boards and the inability to reconstruct the graph in case a board dies.

6.2 Future Work

In future work, there are several areas that could be improved to enhance the functionality, security and efficiency of the network. One approach could involve adding a unique MAC address to each node in the network, which would improve the security and efficiency by providing a more accurate way to locate senders and receivers. That would also mean that the node discovery would take place in each node in the network. If the synchronization of the RTC could be achieved using the network consensus, the use of a coordinator board (master) would become obsolete. The wake-on-radio

function could also be implemented, to simplify the implementation process and prolong the network's lifetime, by allowing nodes to remain in sleep mode until they receive a wake-up call. Another area of focus could be the packet size, that is currently constant, due to the restricted time frame of this thesis. The packet size can become variable, enabling the transmission and reception of different-sized packets and limiting the over-the-air messages and power consumption of the boards.

In addition, the network's multi-protocol feature could be leveraged by adding a Bluetooth stack to provide a more user-friendly interface for monitoring and controlling the network, using smartphones. For the *outdoor demo*, expanding the network, to include a larger number of boards and sensors, would allow the collection of more data and cover a wider range of monitored area. Finally, the network (indoor centralized and outdoor) could be redesigned to be powered only by ambient energy sources (for the master node). Thus, increasing its autonomy and suitability for deployment in remote or hard-to-reach areas, where constant access to power sources may be limited. In this scenario, the use of the non volatile memory, for storing important parameters of the boards, is advisable to be implemented, for robustness purposes. Also, the incorporation of the peltier devices, to the indoor centralized demo, is appropriate.

By implementing these improvements, the network's functionality, security and efficiency would be significantly enhanced, and the network's capabilities would be expanded, making it more versatile and accessible to a broader range of users.

Bibliography

[1] G. Vougioukas, S.-N. Daskalakis, and A. Bletsas, "Could battery-less scatter radio tags achieve 270-meter range?" In *2016 IEEE Wireless Power Transfer Conference (WPTC)*, 2016, pp. 1–3. DOI: 10.1109/WPT.2016.7498843.

[2] P. Vasilakopoulos, "Design and implementation of a low-cost bidirectional embedded backscatter link," Undergraduate Thesis, Technical University of Crete, Chania Greece, 2021. DOI: 10.26233/heallink.tuc.89916.

[3] K. Tountas, "Performance analysis, middleware and hardware for bistatic, ultra-low power scatter radio networks," Undergraduate Thesis, Technical University of Crete, Chania Greece, 2016. DOI: 10.26233/heallink.tuc.66155.

[4] T. Ojha, S. Misra, and N. S. Raghuwanshi, "Wireless sensor networks for agriculture: The state-of-the-art in practice and future challenges," *Computers and Electronics in Agriculture*, vol. 118, pp. 66–84, 2015, ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2015.08.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169915002379.

[5] *Ug309: Thunderboard sense 2 user's guide*, Available at https://www.silabs.com/documents/public/user-guides/ug309-sltb004a-user-guide.pdf (2023/02/02).

[6] *Simplicity studio software*, Available at https://www.silabs.com/developers/simplicity-studio (2023/02/02).

[7] *Efm32™ 32-bit mcu ultra efficient energy modes*, Available at https://www.silabs.com/mcu/32-bit-microcontrollers/efm32-energy-modes (2023/02/02).

[8] *Power manager*, Available at https://docs.silabs.com/gecko-platform/latest/service/power_manager/overview#power-manager (2023/02/02).

[9] *Ruggedized electrical double layer energy storage capacitors up to 3 v operating voltage*, Available at https://gr.mouser.com/datasheet/2/427/235edlc_hvr_enycap-1762949.pdf (2023/02/02).

[10] *Soil moisture*, Available at https://www.drought.gov/topics/soil-moisture#:~:text=As%20defined%20by%20the%20AMS,the%20pores%20of%20the%20soil. (2023/02/02).

[11] S.-N. Daskalakis, "Environmental scatter radio sensors with rf energy harvesting," M.S. thesis, School of Electrical and Computer Engineering, Technical Univercity of Crete, Chania Greece, 1996. DOI: https://doi.org/10.26233/heallink.tuc.66156.

[12] V. Papageorgiou, A. Nichoritis, P. Vasilakopoulos, G. Vougioukas, and A. Bletsas, "Towards Ambiently Powered Inference on Wireless Sensor Networks: Asynchrony is the Key!" 5th IEEE International Workshop on Wireless Communications and Networking in Extreme Environments (WCNEE), Jul. 2021.

[13] W. H. Zhu, Y. Zhu, Z. Davis, and B. J. Tatarchuk, "Energy efficiency and capacity retention of ni–mh batteries for storage applications," *Applied Energy*, vol. 106, pp. 307–313, 2013, ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2012.12.025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261912009075.

[14] H. D. Kambezidis, "The solar radiation climate of greece," *Climate*, vol. 9, no. 12, 2021, ISSN: 2225-1154. DOI: 10.3390/cli9120183. [Online]. Available: https://www.mdpi.com/2225-1154/9/12/183.

[15] *Peltier element tec1-12706*, Available at https://www.mikroe.com/peltier-element-tec1-12706 (2023/02/23).

[16] *Radio abstraction interface layer (rail) sdk*, Available at https://www.silabs.com/developers/flex-sdk-radio-abstraction-interface-layer (2023/02/23).

[17] B. Watson, "Fsk: Signals and demodulation," *Watkins–Johnson Company Tech–notes*, vol. 7, no. 5, 1980. [Online]. Available: https://www.rfcafe.com/references/articles/wj-tech-notes/fsk-signals-demodulation-v7-5.pdf.

[18] *Operation of wireless microphones*, Available at https://www.fcc.gov/consumers/guides/operation-wireless-microphones#:~:text=Unlicensed%20wireless%20microphone%20use%20is,for%20each%20of%20those%20bands. (2023/02/02).

[19] A. Bletsas, *Tel303slides12*, University Lecture, 2013.

[20] *Rail tutorial 4: Combining transmit and receive*, Available at https://community.silabs.com/s/article/rail-tutorial-4-combining-transmit-and-receive?language=en_US (2023/02/02).

[21] *Token-ring networks*, Available at https://www.ibm.com/docs/en/i/7.1?topic=standards-token-ring-networks (2023/02/02).

[22] *Nvm3 - nvm data manager*, Available at https://docs.silabs.com/gecko-platform/3.0/driver/api/group-nvm3 (2023/02/02).

[23] *Gecko platform*, Available at https://docs.silabs.com/gecko-platform/3.0/index (2023/02/02).

[24] *Rail tutorial: Timer synchronization and sleep (power manager)*, Available at https://community.silabs.com/s/article/SSv5-timer-synchronization-and-sleep?language=en_US (2023/02/02).

[25] J. Yoo, S. Jo, and U. Kang, "Supervised belief propagation: Scalable supervised inference on attributed networks," in *2017 IEEE International Conference on Data Mining (ICDM)*, 2017, pp. 595–604. DOI: 10.1109/ICDM.2017.69.

[26] S. Jo, J. Yoo, and U. Kang, "Fast and scalable distributed loopy belief propagation on real-world graphs," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, ser. WSDM '18, Marina Del Rey, CA, USA: Association for Computing Machinery, 2018, pp. 297–305, ISBN: 9781450355810. DOI: 10.1145/3159652.3159722.

[27] P. Norman and L. Jt, "Putting iterative proportional fitting on the researchers desk," Feb. 2000. [Online]. Available: https://www.researchgate.net/publication/2500100_Putting_Iterative_Proportional_Fitting_on_the_Researchers_Desk.

[28] K. Müller and K. Axhausen, "Population synthesis for microsimulation: State of the art," Sep. 2010. [Online]. Available: https://www.researchgate.net/publication/228973867_Population_synthesis_for_microsimulation_State_of_the_art.

[29] J. Mooij and H. Kappen, "Sufficient conditions for convergence of loopy belief propagation," 2012. DOI: 10.48550/arXiv.1207.1405.

[30] L. Xiao, S. Boyd, and S.-J. Kim, "Distributed average consensus with least-mean-square deviation," *Journal of Parallel and Distributed Computing*, vol. 67, no. 1, pp. 33–46, 2007, ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2006.08.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731506001808.

[31] *Rail tutorial: Multi-phy usecases and examples*, Available at https://community.silabs.com/s/article/rail-tutorial-multi-phy-usecases-and-examples?language=en_US (2023/02/02).

[32] *Command line interface*, Available at https://docs.silabs.com/gecko-platform/3.0/service/api/group-cli (2023/02/02).

[33] *Unity*, Available at https://unity.com/ (2023/02/02).

[34] *Ardity*, Available at https://ardity.dwilches.com/ (2023/02/02).